API Authentication using Passport JS and JSON Web Tokens.

-By Himanshu Singh

Prerequisites:

1. Node JS
2. MongoDB
3. JSON Web Token Package
4. Passport package

What is JSON Web Token?

JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. Signed tokens can verify the *integrity* of the claims contained within it, while encrypted tokens *hide* those claims from other parties. When tokens are signed using public/private key pairs, the signature also certifies that only the party holding the private key is the one that signed it.

When should you use JSON Web Tokens?

Authentication: This is the most common scenario for using JWT. Once the user is logged in, each subsequent request will include the JWT, allowing the user to access routes, services, and resources that are permitted with that token.

What is the JSON Web Token structure?

In its compact form, JSON Web Tokens consist of three parts separated by dots (.),

which are:

- Header
- Payload
- Signature

Therefore, a JWT typically looks like the following:

xxx.yyy.zzz

Let's break down the different parts.

Header

The header *typically* consists of two parts: the type of the token, which is JWT, and the hashing algorithm being used, such as HMAC SHA256 or RSA. Then, this JSON is **Base64Url** encoded to form the first part of the JWT.

Payload

The second part of the token is the payload, which contains the claims, session expiry and the response data along with it which you chose to send to the Client Side. Claims are statements about an entity (typically, the user) and additional metadata. Therefore, a jwt_payload typically looks like the following:

```
{ data:
  { _id: '5af93e207b2eb119f8f24e94',
    firstname: 'Mib',
 email: 'mibfanclub2018@gmail.com',
    isAdmin: true },
 iat: 1526553453,
 exp: 1526617453 }
```

Signature:

The signature is used to verify the message wasn't changed along the way, and, in the case of tokens signed with a private key, it can also verify that the sender of the JWT is who it says it is.

Putting all together

The output is three Base64-URL strings separated by dots that can be easily passed in HTML and HTTP environments, while being more compact when compared to XML-based standards such as SAML.

The following shows a JWT that has the previous header and payload encoded, and it is signed with a secret.

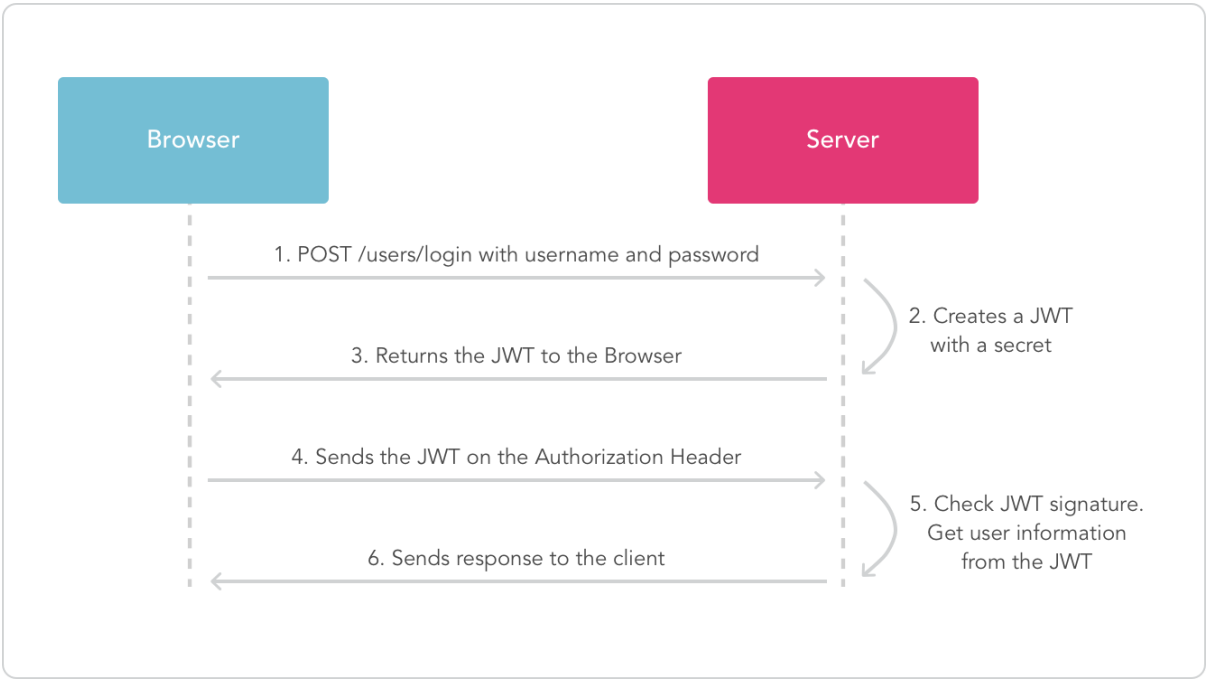
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJkYXRhIjp7Il9pZCI6IjVhZjkzZTIwN2IyZWIxMTlmOGYyNGU5NCIsImZpcnN0bmFtZSI6Ik1pYiIsImVtYWlsIjoibWliZmFuY2x1YjIwMThAZ21haWwuY29tIiwiaXNBZG1pbiI6dHJ1ZX0sImlhdCI6MTUyNjU1MzQ1MywiZXhwIjoxNTI2NjE3NDUzfQ.XSDLMGBeJysmOGXWO8YZCLJZ6Yg1uRDFmHHGekFIRweyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJkYXRhIjp7Il9pZCI6IjVhZjkzZTIwN2IyZWIxMTlmOGYyNGU5NCIsImZpcnN0bmFtZSI6Ik1pYiIsImVtYWlsIjoibWliZmFuY2x1YjIwMThAZ21haWwuY29tIiwiaXNBZG1pbiI6dHJ1ZX0sImlhdCI6MTUyNjU1MzQ1MywiZXhwIjoxNTI2NjE3NDUzfQ.XSDLMGBeJysmOGXWO8YZCLJZ6Yg1uRDFmHHG-ekFIRw

How do JSON Web Tokens work?

In authentication, when the user successfully logs in using their credentials, a JSON Web Token will be returned and must be saved locally (typically in local storage, but cookies can be also used), instead of the traditional approach of creating a session in the server and returning a cookie.

Whenever the user wants to access a protected route or resource, the user agent should send the JWT, typically in the **Authorization** header using localStorage. The content of the header should look like the following:

```
 headers: new HttpHeaders({ 'Authorization':
localStorage.getItem('jwtToken')  })
```



Follow the following steps for authenticating your api with passport using jsonwebtoken:

Step 1:(installing dependencies.)

npm install passport jsonwebtoken –save

Step 2:(require dependencies in your server.js file.)

> In this file, we bring in all the modules required for our application like express, passport, jsonwebtoken, mongo, MongoClient, db and etc. We create the database connection using MongoClient.*connect()*.

```
const jwt = require('jsonwebtoken');
const passport = require('passport');
const mongo = require("mongodb");
const MongoClient = require("mongodb").MongoClient;
const db = require("./config/db");
```

Step 3:(initialize passport middleware in your server.js file)

app.use(passport.initialize());
app.use(passport.session());

In Express-based application, passport.initialize() middleware is required to initialize Passport. If your application uses persistent login sessions, passport.session() middleware must also be used.

Step 4:(pass database and mongo module)

You have to pass database and required mongo module from your server.js file to passport.js file in the config folder.

To pass write the below code while setting connection of MongoClient to database.

```
require('./config/passport')(passport, database, mongo);
```

Or Alternatively:(Not recommended)

You have to again require both the database and mongo module again in your passport.js file.

```
const mongo = require("mongodb");
const db = require("./config/db");
```

Step 5:(configure passport.js file)

In this part, we initialize the passport.js and define strategies. In our config folder, we create passport.js file where we define our JWT Strategy.

```
1  const JwtStrategy = require('passport-jwt').Strategy;
2  const ExtractJwt = require('passport-jwt').ExtractJwt;
3
4  module.exports = function(passport, db, mongo) {
5      var opts = {}
6      opts.jwtFromRequest = ExtractJwt.fromAuthHeaderAsBearerToken();
7      opts.secretOrKey = 'MYSECRETKEY';
8      console.log(opts);
9      passport.use(new JwtStrategy(opts, (jwt_payload, done) => {
10         console.log(jwt_payload.data._id);
11         var objectId = new mongo.ObjectId(jwt_payload.data._id);
12         db.collection("user").findOne({ "_id": objectId }, (err, user) => {
13             if (err) {
14
15                 return done(err, false);
16             }
17             if (user) {
18                 return done(null, user);
19             } else {
20                 return done(null, false);
21             }
22         });
```

We use "*passport.use(new **JwtStrategy**(opts, (jwt_payload, done) => {*" which defines our JWT strategy. The callback includes payload, which includes user information which we set in login route while generating json web token when a user login . Try consoling payload and opts to explore more.

Step 6:(*we start generating token on /login route*)

```
app.post("/login", function(req, res) {


    var myquery = { email: req.body.email.toLowerCase() };
    db.collection("user").findOne(
        myquery,
        (err, result) => {
            if (err) {
                console.log('err in if');
            }
            if (result) {
                // console.log(result);
                if (verifyPassword(req.body.pswd, result.password)) {
                    let responseData = {
                        _id: result._id,
                        firstname: result.firstname,
                        email: result.email,
                        isAdmin: result.isAdmin
                    };
                    var token = jwt.sign({ data: responseData }, 'MYSECRETKEY', { expiresIn: 64000 });
                    res.send({
                        data: responseData,
                        token: `Bearer ${token}`
                    });
                } else {
                    res.send(false);
                }

            } else {
                console.log('not found');
                res.send(false);
            }
        });
```

We start by getting the email and finding it in database :
var query = {email : req.body.email}

1. The verifyPassword function accepts the submitted password and hashed password in database as arguments, to match it. If the password does not matches, we console “not found” message and send response as false.
2. If the email and password is found, now , JWT Token is generated(using jwt.sign() method”).
3. We can Store the JWT token in the client side in cookies or local storage and request to protected route to include the token within the header.

Step 7:(Testing JWT Token in header using route protection)

To test out our JWT token with header, we first have to protect our route by inlining the following code in user.js file.

In the backend(Node):

```
app.get("/getUserData/:email", passport.authenticate('jwt', {
session: false }), function(req, res) {
```

In the Frontend(Node):

Add the authorization in the header while making an api call

```
  headers: new HttpHeaders({
        'Authorization': localStorage.getItem('jwtToken')
      })
```